AFRL-IF-RS-TR-2000-60
Final Technical Report
April 2000

ADVANCED COMPILER TECHNOLOGY FOR SCALABLE PARALLEL MACHINES

Computer Systems Laboratory

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. D515

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-60 has been reviewed and is approved for publication.

APPROVED: Joseph A Carozzoni

JOSEPH A. CAROZZONI
Project Engineer

FOR THE DIRECTOR: Northrup Fowler

NORTHRUP FOWLER, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

ADVANCED COMPILER TECHNOLOGY FOR SCALABLE PARALLEL MACHINES

Monica Lam

Contractor: Computer Systems Laboratory
Contract Number: F30602-95-C-0098
Effective Date of Contract: 31 May 1995
Contract Expiration Date: 28 February 1999
Program Code Number: C276
Short Title of Work: Advanced Compiler Technology For Scalable Parallel
Period of Work Covered: May 95 – Feb 99

Principal Investigator: Monica Lam
Phone: (650) 725-3927
AFRL Project Engineer: Joseph A. Carozzoni
Phone: (315) 330-7796

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Joseph A. Carozzoni, AFRL/IFTB, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2000	Final May 95 - Feb 99

4. TITLE AND SUBTITLE
ADVANCED COMPILER TECHNOLOGY FOR SCALABLE PARALLEL MACHINES

5. FUNDING NUMBERS
C - F30602-95-C-0098
PE - 62301E
PR - C276
TA - 00
WU - 01

6. AUTHOR(S)
Monica Lam

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Systems Laboratory
Stanford University
Stanford CA 94305

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFTB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-60

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Projects Engineer: Joseph A. Carozzoni/IFTB/(315) 330-7796

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
This report presents an extensive empirical evaluation of an interprocedural parallelizing compiler, developed as part of the Stanford SUIF compiler system. The system incorporates a comprehensive and integrated collection of analysis, including privatization and reduction recognition for both array and scalar variables, and symbolic analysis of array subscripts. The interprocedural analysis framework is designed to provide analysis results nearly as precise as full inlining but without its associated costs. Experimentation with this system shows that it is capable of detecting coarser granularity of parallelism then previously possible. Specifically, it can parallelize loops that span numerous procedures and hundreds of lines of codes, frequently requiring modifications to array data structures such as privatization and reduction transformations. Measurements from several standard benchmark suites demonstrate that an integrated combination of interprocedural analysis can substantially advance the capabilities of automatic parallelization technology.

14. SUBJECT TERMS
Software, Knowledge-Based Systems

15. NUMBER OF PAGES
20

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1 Executive Summary

The SUIF parallelizing compiler research project has made several major contributions to the field of compiler techniques for high-performance computing. We have developed a large suite of new compiler techniques that are proven to be critical to effective parallelization: they include interprocedural analysis techniques for finding coarse-grain parallelism, affine program and data transforms to improve the memory subsystem performance, and a program analysis for disambiguating pointer variables which is a prerequisite to parallelizing any C programs. We have also improved the interaction between the compiler and the operating system to further enhance the memory subsystem performance. All the compiler techniques have been implemented in the SUIF parallelizing compiler, which has been demonstrated to parallelize many more programs effectively when compared to state-of-the-art commercial compilers. We have also developed an interactive parallelization system called the SUIF Explorer, which guides the user in providing additional information to improve the parallelization. Finally, we have developed a run-time system that simplifies the programming of networks of workstations by providing the programmer with the abstraction of a global object space as well as fault tolerance.

The compiler techniques we developed have been implemented in several commercial compilers such as the SGI and Intel compilers. We have made the SUIF compiler infrastructure publicly available, and it has been used by many different institutions for research on a large number of different topics. These topics include scalar optimizations, parallelization, memory optimizations, interprocedural analysis, code specialization, partial evaluation, profile-driven optimization, code generation, VLIW machines, vector machines, multimedia processors, digital signal processors, embedded RAM processors, and reconfigurable hardware. Universities using the compiler system for research or teaching purposes include Aachen University of Technology, Colorado State University, Dresden University of Technology, Georgia Institute of Technology, Harvard University, IIE-CNAM, Institut National Polytechnique de Grenoble, MIT, UC Berkeley, Oregon Graduate Institute, Princeton University, Seoul National University, Stanford University, University of Adelaide, UC Santa Barbara, University of Cincinnati, University of Delaware, University of Manitoba, University of Maryland, University of Michigan, University of Minnesota, University of Sussex, University of Toronto, University of Queensland, University of Wisconsin, Wayne State University, and Yale University. Companies and research institutions that use SUIF include IRISA/INRIA, Synopsys Inc and USC/ISI. The SUIF compiler has been selected by DARPA and NSF as a basis of the National Compiler Infrastructure system.

The project has also produced a large number of Ph.D. graduates and furthered the education of a number of post-doctorates. The names and current job positions of the members of the research project are: Saman Amarasinghe, Assistant Professor at MIT, Jennifer Anderson, Researcher at Compaq Western Research Laboratories, Amer Diwan, Assistant Professor at University of Colorado, Mary Hall, Researcher at USC/ISI, Martin Rinard, Assistant Professor at MIT, Daniel Scales, Researcher at Compaq Western Research Laboratories, Chau-Wen Tseng, Assistant Professor at University of Maryland, and Robert Wilson, Member of Technical Staff at Tensilica Inc.

2 Interprocedural analysis for parallelization

Existing commercially available parallelizing compilers are not effective at getting good performance on multiprocessors. As these parallelizers were developed from vectorizing compiler technology, they tend to be successful in parallelizing only innermost loops. Parallelizing just inner loops is not adequate for multiprocessors for two reasons. First, inner loops may not make up a significant portion of the sequential computation, thus limiting the parallel speedup by limiting the amount of parallelism. Second, synchronizing processors at the end of the inner loops leaves little computation occurring in parallel between synchronization points. The cost of frequent synchronization and load imbalance can potentially overwhelm the benefits of parallelization.

We have developed an automatic parallelization system that is fully interprocedural[7, 1, 6]. The system incorporates all the standard analyses included in today's automatic parallelizers, such as data dependence analysis, analyses of scalar variables including scalar constant propagation, value numbering, induction variable recognition, scalar privatization scalar dependence and reduction recognition. In addition, the system employs analyses for array privatization and array reduction recognition. The implementation of these techniques extends previous work to meet the demands of parallelizing real programs. The interprocedural analysis is designed to be practical while providing nearly the same quality of analysis as if the program were fully inlined. Our system has been shown to be capable of finding parallelism in codes spanning over a thousand lines of code and many different procedures.

We have demonstrated that interprocedural array data-flow analysis, array privatization, and reduction recognition are key technologies that greatly improve the success of automatic parallelization. By finding coarse-grain parallelism, the compiler increases parallelization coverage, lowers synchronization costs and improves speedups. Through our work, we discovered that the effectiveness of an interprocedural parallelization system depends on the strength of all the individual analyses, and their ability to work together in an integrated fashion. This comprehensive approach to parallelization analysis is why our system has been so much more effective at automatic parallelization than previous interprocedural systems and commercially available compilers.

3 Affine Program Transform

We have developed a new affine framework and algorithms to improve parallelism and locality[10, 11, 12]. An affine partitioning framework unifies many useful program transforms such as unimodular transformations (interchange, reversal, skewing), loop fusion, fission, scaling, reindexing, and statement reordering.

Based on this framework, we have developed an algorithm that maximizes parallelism while minimizing communication in programs with arbitrary loop nestings and affine data accesses. Our algorithm can find the optimal affine partition that maximizes the degree of parallelism with the minimum degree of synchronizations. In addition, it uses a greedy algorithm to minimize communication between loops heuristically by aligning the computation partitions for different loops, trading off excess degrees of parallelism, and choosing pipelined parallelism over doall parallelism if it can significantly reduce the communication. The algorithm is optimal in maximizing the degrees of parallelism that require (1) no communication, (2) near-neighbor communication and a constant number of synchronizations, and (3) near-neighbor communication and $O(n)$ synchronizations where n is the number of iterations in a loop.

Our algorithm subsumes previous work that uses loop transforms (unimodular, loop fusion, fission, scaling, reindexing, and statement reordering) as well as previous data and computation distribution and barrier synchronization elimination algorithms.

4 Automatic Data Transform

Effective memory hierarchy utilization is critical to the performance of modern multiprocessor architectures. We have developed the first compiler system that fully automatically parallelizes sequential programs and changes the original array layouts to improve memory system performance[4, 3]. Our optimization algorithm consists of two steps. The first step chooses the parallelization and computation assignment such that

synchronization and data sharing are minimized. The second step then restructures the layout of the data in the shared address space with an algorithm that is based on a new data transformation framework. We ran our compiler on a set of application programs and measured their performance on the Stanford DASH multiprocessor. Our results show that the compiler can effectively optimize parallelism in conjunction with memory subsystem performance.

This work is useful for purposes other than translating sequential code to shared memory multiprocessors. Our algorithm to determine how to parallelize and distribute the computation and data is useful also to distributed address space machines. Our data transformation framework, consisting of the strip-mining and permuting primitives, is applicable to layout optimization for uniprocessors. Finally, our data transformation algorithm can also apply to HPF programs. While HPF directives are originally intended for distributed address space machines, our algorithm uses the information to make data accessed by each processor contiguous in the shared address space. In this way, the compiler achieves locality of reference, while taking advantage of the cache hardware to provide memory management and coherence functions.

5 Pointer alias analysis

Pointer analysis promises significant benefits for optimizing and parallelizing compilers, yet despite much recent progress it has not advanced beyond the research stage. Several problems remain to be solved before it can become a practical tool. First, the analysis must be efficient without sacrificing the accuracy of the results. Second, pointer analysis algorithms must handle real C programs. If an analysis only provides correct results for well-behaved input programs, it will not be widely used. We have developed a pointer analysis algorithm that addresses these issues.

We have developed a fully context-sensitive pointer analysis algorithm and have shown that it is very efficient for a set of C programs[16, 17]. To make context sensitivity feasible, we have developed the concept of partial transfer functions which minimizes re-analysis of a procedure by capturing the results of the analysis in a parameterized manner for a subset of the domain. The algorithm is based on the simple intuition that the aliases among the inputs to a procedure are the same in most calling contexts. Even though it is difficult to summarize the behavior of a procedure for all inputs, we can find partial transfer functions for the input aliases encountered in the program. This allows us to analyze a procedure once and reuse the results in many other contexts.

Even though our algorithm is still exponential in the worst case, we have so far found that it performs well. As long as most procedures are always called with the same alias patterns, our algorithm will continue to avoid exponential behavior. To be safe, after reaching some limit on the number of PTFs per procedure, we could easily generalize the PTFs instead of creating new ones.

Our analysis can handle all the features of the C language. We make conservative assumptions where necessary to ensure that our results are safe. Even though we may occasionally lose some precision due to these conservative assumptions, we believe it is important to handle the kinds of code found in real programs, even if they do not strictly conform to the ANSI standard.

6 Interactions with operating systems

We have developed a new technique, compiler-directed page coloring, that eliminates conflict misses in multiprocessor applications[5]. It enables applications to make better use of the increased aggregate cache size available in a multiprocessor. This technique uses the compiler's knowledge of the access patterns of the parallelized applications to direct the operating system's virtual memory page mapping strategy. We demonstrate that this technique can lead to significant performance improvements over two commonly used page mapping strategies for machines with either direct-mapped or two-way set-associative caches. We also show that it is complementary to latency-hiding techniques such as prefetching.

We implemented compiler-directed page coloring in the SUIF parallelizing compiler and on two commercial operating systems. We applied the technique to the SPEC95fp benchmark suite, a representative set of numeric programs. We used the SimOS machine simulator to analyze the applications and isolate their

performance bottlenecks. We also validated these results on a real machine, an eight-processor 350MHz Digital AlphaServer. Compiler-directed page coloring leads to significant performance improvements for several applications. Overall, our technique improves the SPEC95fp rating for eight processors by 8% over Digital UNIX's page mapping policy and by 20% over a page coloring, a standard page mapping policy. The SUIF compiler achieves a SPEC95fp ratio of 63.84 on an 8-processor 440Mhz AlphaServer, which was the highest ratio at that time.

7 Evaluation of compiler on whole applications

All the compiler techniques developed have been implemented in the SUIF compiler system. The system can find coarse-grain parallel loops previously not found by any other automatic systems because of its large collection of advanced interprocedural parallelization analysis. Moreover, it is able to get high absolute performance out of the parallel code because of its high-level data and loop transformations to improve memory subsystem performance. These techniques have a significant impact on the performance of half of the NAS and SPECfp95 benchmark suites. It outperforms the state-of-the-art commercial compiler by 50% in parallelizing the SPEC95fp programs[2, 8].

For some programs, our analysis is sufficient to find the available parallelism. For other programs, it seems impossible or unlikely that a purely static analysis could discover parallelism—either because correct parallelization requires dynamic information not available at compile time or because it is too difficult to analyze. In such cases, we can benefit from some support for run-time parallelization or user interaction. The aggressive static parallelizer we have built provides a good starting point to investigate these techniques.

8 The SUIF Explorer: An Interactive Parallelizer

While the interprocedural parallelization analysis can find some coarse-grain loops to parallelize, the procedure, however, is fragile, as a single dependence in a large, otherwise parallel, loop can ruin the program's parallel performance. Even a compiler that included every single conceivable parallelization technique would be inadequate, because compilers are fundamentally limited by the sequential semantics originally coded into the program. Often times, it requires application-specific knowledge to modify the algorithm to make it parallelizable. The SUIF Explorer is an interactive parallelization tool that guides the user in supplying additional information so as to extend the capability of the interprocedural analysis[9].

The SUIF Explorer is more effective than previous systems in minimizing the number of lines of code that require programmer assistance. First, the interprocedural analyses in the SUIF system is successful in parallelizing many coarse-grain loops, thus minimizing the number of spurious dependences requiring attention. We found that the dependences left unresolved by the SUIF parallelizer are mostly nontrivial and are deserving of human attention. Second, the system uses dynamic execution analyzers to identify those important loops that are likely to be parallelizable. This greatly reduces the number of loops that the programmer needs to pay attention to. Third, the SUIF Explorer is the first to apply program slicing to aid programmers in interactive parallelization. Slicing reduces the number of lines that need to be analyzed and minimizes the likelihood for human error. The system guides the programmer in the parallelization process using a set of sophisticated visualization techniques.

Our experience with the prototype SUIF Explorer system suggests that the tool can be very effective in improving the parallel performance of large applications. We demonstrate the effectiveness on three programs. The MDG benchmark improves from no speedup at all to a 6-times speedup on 8 processors, Arc3d improves from 1.6 to 4.9 and Hydro improves from 2.7 to 4.3.

9 Shared address memory system for networks of heterogeneous workstations

Distributed memory systems, especially in the form of networks of workstations, are an important computational resource. However, programming distributed memory machines using commonly available message-

passing libraries is a difficult process. The difficulties become even greater for very sophisticated scientific applications that have highly irregular parallelism and communication. We have developed a portable run-time system called SAM which provides the user with the abstraction of a global name space with the efficiency of automatic caching of shared data[14]. SAM incorporates mechanisms to address the problem of high communication overheads on distributed memory machines; these mechanisms include tying synchronization to data access, chaotic access to data, prefetching of data, and pushing of data to remote processors.

We found that the performance of our SAM applications depends fundamentally on the scalability of the underlying parallel algorithm, and whether the algorithm's communication requirements can be satisfied by the hardware. Our experience suggests that SAM is successful in allowing programmers to use distributed memory machines effectively with much less programming effort than required previously.

The SAM system also provides fault tolerance, which is important for large-scale systems and in environments where application users do not have absolute control. SAM supports fault tolerance efficiently by ensuring that data are replicated on more than one workstation using the dynamic caching already provided by SAM. Our method is efficient as it avoids expensive writes to disk and does not require a common file server, and each process checkpoints independently and only when sending data which is not reproducible to another process[13].

References

[1] Saman P. Amarasinghe. *Parallelizing Compiler Techniques Based on Linear Inequalities.* PhD thesis, Stanford University, January 1997.

[2] Saman P. Amarasinghe, Jennifer M. Anderson, Chris S. Wilson, Shih-Wei Liao, Brian M. Murphy, Robert S. French, Monica S. Lam, and Mary W. Hall. Multiprocessors from a software perspective. *IEEE Micro*, pages 52–61, June 1996. A special issue on the Hot Chips VII Conference, Stanford, CA, Aug. 1995.

[3] Jennifer M. Anderson. *Automatic Computation and Data Decomposition for Mulltiprocessors.* PhD thesis, Stanford University, March 1997.

[4] Jennifer M. Anderson, Saman P. Amarasinghe, and Monica S. Lam. Data and computation transformations for multiprocessors. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 166–178, Santa Barbara, CA, July 1995.

[5] Edouard Bugnion, Jennifer M. Anderson, Todd C. Mowry, Mendel Rosenblum, and Monica S. Lam. Compiler-directed page coloring for multiprocessors. In *Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, October 1996.

[6] Mary W. Hall, Saman P. Amarasinghe, Brian R. Murphy, Shih-Wei Liao, and Monica S. Lam. Detecting coarse-grain parallelism using an interprocedural parallelizing compiler. In *Supercomputing '95*, December 1995.

[7] Mary W. Hall, Saman P. Amarasinghe, Brian R. Murphy, Shih-Wei Liao, and Monica S. Lam. A fully interprocedural system for automatic parallelization. In *Proc. 8th Workshop on Languages and Compilers for Parallel Computing*, Columbus, Ohio, August 1995.

[8] Mary W. Hall, Jennifer M. Anderson, Saman P. Amarasinghe, Brian R. Murphy, Shih-Wei Liao, Edouard Bugnion, and Monica S. Lam. Maximizing performance on multiprocessors with the SUIF compiler. *Computer*, December 1996. A special issue on multiprocessors.

[9] Shih-Wei Liao, Amer Diwan, Jr. Robert P. Bosch, Anwar Ghuloum, and Monica S. Lam. Suif explorer: An interactive and interprocedural parallelizer. In *Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1999.

[10] A. W. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine transforms. In *Conference Record of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, January 1997.

[11] A. W. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine partitions. *Parallel Computing*, 24(3–4):445–475, May 1998.

[12] Amy. W. Lim, Gerald I. Cheong, and Monica S. Lam. An affine partitioning algorithm to maximize parallelism and minimize communication. In *Proceedings of the 13th ACM SIGARCH International Conference on Supercomputing*, Rhodes, Greece, June 1999.

[13] D. J. Scales and M. S. Lam. Transparent fault tolerance for parallel applications on networks of workstations. In *Proceedings of the 1996 USENIX Technical Conference*, San Diego, CA, January 1996.

[14] Daniel J. Scales. *The Design and Evaluation of an Efficient Shared Object System for Distributed Memory Machines.* PhD thesis, Stanford University, December 1995.

[15] Chau-Wen Tseng, Jennifer M. Anderson, Saman P. Amarasinghe, and Monica S. Lam. Unified compilation techniques for shared and distributed address space machines. In *Proceedings of the 1995 International Conference on Supercomputing*, pages 67–76, Barcelona, Spain, July 1995.

[16] Robert P. Wilson. *Efficient Context-Sensitive Pointer Analysis For C Programs.* PhD thesis, Stanford University, December 1997.

[17] Robert P. Wilson and Monica S. Lam. Efficient context-sensitive pointer analysis for c programs. In *Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation*, La Jolla, CA, June 1995.

JOSEPH A. CAROZZONI AFRL/IFTB 525 BROOKS ROAD ROME, NY 13441-4505	2	
COMPUTER SYSTEMS LABORATORY STANFORD UNIVERSITY STANFORD,CA 94305	1	
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1	2
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 3725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	1	4
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1	7
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1	12
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1	17
AFRL/MLME 2977 P STREET, STE 6 WRIGHT-PATTERSON AFB OH 45433-7739	1	19
AFRL/HESC-TDC 2698 G STREET, BLDG 190 WRIGHT-PATTERSON AFB OH 45433-7604	1	22
ATTN: SMDC IM PL US ARMY SPACE & MISSILE DEF CMD P.O. BOX 1500 HUNTSVILLE AL 35807-3801	1	24

TECHNICAL LIBRARY D0274(PL-TS) SPAWARSYSCEN 53560 HULL ST. SAN DIEGO CA 92152-5001	1	26
COMMANDER, CODE 4TL000D TECHNICAL LIBRARY, NAWC-WD 1 ADMINISTRATION CIRCLE CHINA LAKE CA 93555-6100	1	27
CDR, US ARMY AVIATION & MISSILE CMD REDSTONE SCIENTIFIC INFORMATION CTR ATTN: AMSAM-RD-OB-R, (DOCUMENTS) REDSTONE ARSENAL AL 35898-5000	2	31
REPORT LIBRARY MS P364 LOS ALAMOS NATIONAL LABORATORY LOS ALAMOS NM 87545	1	33
AFIWC/MSY 102 HALL BLVD, STE 315 SAN ANTONIO TX 78243-7016	1	38
ATTN: KAROLA M. YOURISON SOFTWARE ENGINEERING INSTITUTE 4500 FIFTH AVENUE PITTSBURGH PA 15213	1	39
USAF/AIR FORCE RESEARCH LABORATORY AFRL/VSOSA(LIBRARY-BLDG 1103) 5 WRIGHT DRIVE HANSCOM AFB MA 01731-3004	1	44
ATTN: EILEEN LADUKE/D460 MITRE CORPORATION 202 BURLINGTON RD BEDFORD MA 01730	1	45
DUSD(P)/DTSA/DUTD ATTN: PATRICK G. SULLIVAN, JR. 400 ARMY NAVY DRIVE SUITE 300 ARLINGTON VA 22202	1	46
AFRL/IFT 525 BROOKS ROAD ROME, NY 13441-4505	1	1365

AFRL/IFTM
525 BROOKS ROAD
ROME, NY 13441-4505

1 1866

*Total Number of Copies is: 24

I have verified that this address list is correct and complete. I Have also checked the mailing labels to see that they are correct and are ready for use.

Joseph A Camuzzone

Signature